对 Boston 数据集的降维分析

CS245　数据科学基础　陆朝俊
叶泽林 515030910468

1　问题描述

在数据预处理中，数据约简是一个重要的步骤，数据约简技术可以得到数据集的约简表示，即缩小数据容量但保持了原始数据的大多数信息，使得之后的分析更加高效，而分析结果与未约简的结果几乎相同。

在数据量和数据复杂性日益增多情况下，数据约简更是数据预处理中不可或缺的关键一环。这次作业中，我将使用数据约简技术（以 PCA 为主）对 Boston 数据集进行降维分析。

2　解决方案[1]

2.1　数据集获取及读入

Boston 数据集的全称为波士顿房价数据集（Boston House Price Dataset），给定房屋及其相邻房屋的详细信息进行房价预测，是一个针对回归问题的数据集。该数据集包含 506 个实例，每个实例拥有 13 个特征及一个回归目标（房价），具体的数据集特征情况可参见附录 A.1。

Boston 数据集已集成在 Python 的 scikit-learn 模块下，安装好该模块后只需运行以下代码即可读取数据集：

```
1 from sklearn import datasets
2 boston = datasets.load_ boston()
```

2.2　降维分析

降维分析中最常用的就是主成分分析（PCA）算法，本次作业中也将采用 PCA 算法进行降维分析。PCA 的主要思想是：找出能反映最大偏差的特征的线性组合（即主成分），构成新特征空间。

PCA 算法中主成分维数的选择将影响预处理后的分析过程：维数过少，无法获得有效的特征；维数过多，数据约简的作用将被弱化。因此，这次作业的降维分析分为如下步骤：

1. 读入数据集；
2. 以 1 至 13（原特征总数）为主成分维数，执行 PCA 算法，并统计各维数下主成分的方差占比之和以及 PCA 算法运行所消耗的时间；
3. 推断 Boston 数据集主成分的最佳维数；
4. 将主成分为 1 至 3 的情况可视化，对比检查其降维效果。

[1]本次作业的所有代码实现可参见附录 A.3

3 结果展示

3.1 PCA 算法在不同主成分维数下对 Boston 数据集的降维效果

一般地，PCA 算法得到的主成分越多，其方差占比之和也就越高，即包含了越多的原数据特征信息，但具体的数值以及主成分的最佳个数（尽可能保持多的特征信息时最低的个数）需要通过实验才能得出。在这次作业中，我统计了主成分个数从 1 至 13 时的方差占比之和以及 PCA 算法消耗的时间，具体数值可参见表 1。

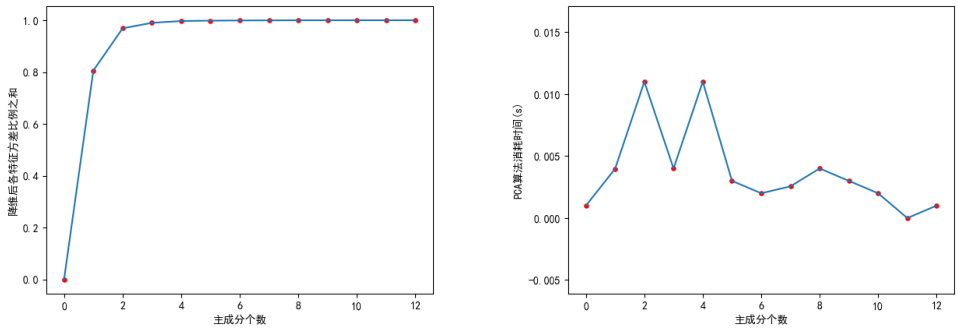
表 1: Boston 数据集 PCA 降维效果比较

主成分个数	主成分方差占比之和	消耗时间 (s)
1	0.80581	0.00300
2	0.96887	0.01001
3	0.99021	0.00600
4	0.99717	0.00901
5	0.99848	0.00400
6	0.99921	0.00400
7	0.99963	0.00400
8	0.99988	0.00300
9	0.99996	0.00400
10	0.99999	0.00400
11	0.99999	0.00100
12	0.99999	0.00100
13	1.00000	0.00000

从表中数值可以看出，主成分个数较少时，方差占比之和随着主成分个数增多有着较大的提升；主成分个数从 3 之后，方差占比之和仅有小幅度的提升，且提升速度越来越慢。

从算法消耗的时间上看，主成分个数较少时，消耗的时间同样随着主成分个数增多而提升较快；当主成分个数接近原特征数时，算法对原数据的更改较少，消耗的时间接近于 0（关于这部分更深入的讨论可参见 3.3 小节。

数值数据虽然精确，但不够直观，从图 1 中可以更直观地看出上述趋势。



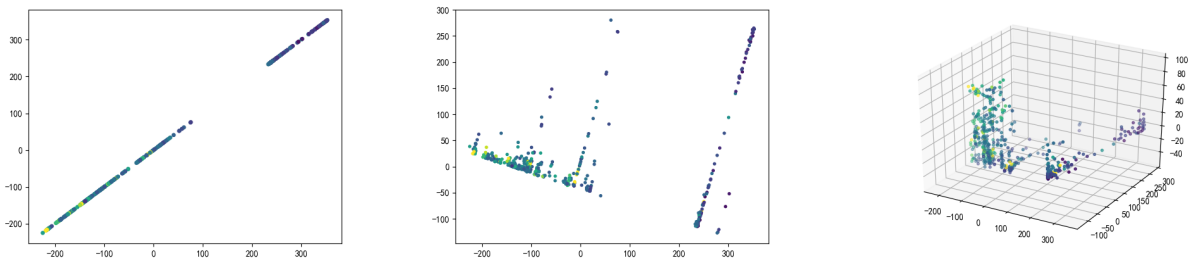
(a) 方差占比之和的变化　　(b) 时间消耗的变化

图 1: PCA 对于 Boston 数据集的降维效果

综合图表数据以及上述分析结果，可以得出：一般应用中，PCA 算法在 Boston 数据集下的最佳主成分个数为 3。

3.2 可视化部分 PCA 效果

为确认 PCA 算法的效果，我将主成分个数为 1 至 3 的结果可视化（图 2）。



(a) 主成分个数为 1　　(b) 主成分个数为 2　　(c) 主成分个数为 3

图 2: PCA 效果可视化（图中不同点的颜色代表不同的回归目标，即房价，颜色越接近则表示数值越接近）

容易看出，PCA 算法在主成分个数为 1 时，仅提取出个数为 2 的部分特征信息（即图 2(b) 右侧的长线）；同理，个数为 2 的结果也仅为图 2(c) 在俯视角度下的信息。因此，上图更为形象地解释了 3.1 小节的结果。

同时，我也对主成分个数为 3 的可视化进行了多角度的探索，详细结果可参见附录 A.2。

3.3 PCA 在实际应用中的思考

本次实验中，我主要根据 PCA 算法的目标主成分个数执行算法，而在实际应用中，有时也需要直接根据主成分方差之和来执行 PCA 算法，尤其是在特征数极多的情况下（如基因芯片中的数万甚至数百万特征）。

Boston 数据集属于极小的数据集，仅有 506 个实例和 13 个特征，因此执行 PCA 算法所消耗的时间可忽略不计，且受噪声的影响较大

A 附录

A.1 Boston 数据集特征信息

表 2: Boston 数据集特征信息

编号	特征名	特征含义
1	CRIM	城镇人均犯罪率
2	ZN	住宅用地超过 25000 sq.ft. 的比例
3	INDUS	城镇非零售商用土地的比例
4	CHAS	查理斯河空变量（如果边界是河流，则为 1；否则为 0）
5	NOX	一氧化氮浓度
6	RM	住宅平均房间数
7	AGE	1940 年之前建成的自用房屋比例
8	DIS	到波士顿五个中心区域的加权距离
9	RAD	辐射性公路的接近指数
10	TAX	每 10000 美元的全值财产税率
11	PTRATIO	城镇师生比例
12	B	1000(Bk-0.63)2，其中 Bk 指代城镇中黑人的比例
13	LSTAT	人口中地位低下者的比例

A.2 PCA 算法在主成分为 3 时的进一步可视化

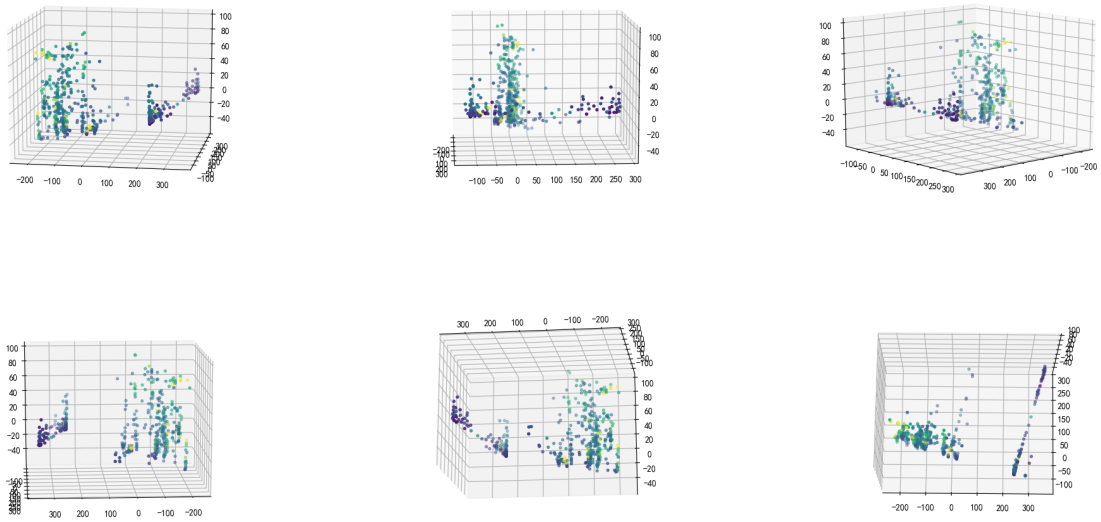


图 3: PCA 效果进一步可视化（主成分为 3 时）

A.3 主要代码

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']     # display Chinese
plt.rcParams['axes.unicode_minus'] = False       # display minus sign
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA

import os
import time

class bostonAnalyzer(object):
    def __init__(self):
        # load dataset
        self.dataset = datasets.load_boston()
        self.data = self.dataset.data
        self.target = self.dataset.target

        # for plot
        self.var = []
        self.t = []

        if not os.path.exists('report/img'):
            os.makedirs('report/img')

    def run(self, vis=False):
        '''
        Run PCA for 1-13 dimensions
        :param vis: True: plot dim 1-3; False: not plot
        '''

        self.var.clear()
        self.t.clear()

        with open('report/result.txt', 'w') as f:
            for i in range(13):
                t = time.time()
                pca_op = PCA(n_components=i)
                pca_res = pca_op.fit_transform(self.data)
                t = time.time() - t

                self.var.append(pca_op.explained_variance_ratio_.sum()
                    )
                self.t.append(t)

                # write log
                f.write('###### Dimension %d ######\n' % i)
                f.write(str(pca_res.shape) + '\n')
```

```
47                  f.write(str(pca_op.explained_variance_ratio_) + '\n')
48                  f.write(str(pca_op.explained_variance_ratio_.sum()) +
                        '\n')
49                  f.write(str(t) + '\n\n')
50
51                  # visualize for debug & plot
52                  if vis:
53                      if i == 1:
54                          # plot 1 dimension
55                          plt.scatter(pca_res[:, 0], pca_res[:, 0], s
                                =14, c=self.target)
56                          plt.savefig('report/img/pca-%d' % i)
57                          plt.show()
58                      elif i == 2:
59                          # plot 2 dimensions
60                          plt.scatter(pca_res[:,0], pca_res[:,1], s=8, c
                                =self.target)
61                          plt.savefig('report/img/pca-%d' % i)
62                          plt.show()
63                      elif i == 3:
64                          # plot 3 dimensions
65                          ax = plt.subplot(projection='3d')
66                          ax.scatter(pca_res[:, 0], pca_res[:, 1],
                                pca_res[:, 2], s=8, c=self.target)
67                          plt.savefig('report/img/pca-%d' % i)
68                          plt.show()
69
70      def show(self):
71          '''
72          Show the basic info of Boston dataset & plot k-lines
73          '''
74          print(self.data.shape)
75          print(self.target.shape)
76          self._k_line_radio()
77          self._k_line_time()
78
79      def _k_line_radio(self):
80          '''
81          Plot k-line of variant radio
82          '''
83          x = list(range(len(self.var)))
84          plt.scatter(x, self.var, s=14, c='r')
85          plt.plot(x, self.var)
86          plt.xlabel('主成分个数')
87          plt.ylabel('降维后各特征方差比例之和')
88          plt.savefig('report/img/kline-radio')
89          plt.show()
90
```

```
91      def _k_line_time(self):
92          '''
93          Plot k-line of time
94          '''
95          x = list(range(len(self.t)))
96          plt.scatter(x, self.t, s=14, c='r')
97          plt.plot(x, self.t)
98          plt.xlabel('主成分个数')
99          plt.ylabel('PCA算法消耗时间(s)')
100         plt.savefig('report/img/kline-time')
101         plt.show()
102
103 if __name__ == '__main__':
104     bA = bostonAnalyzer()
105     bA.run(True)
106     bA.show()
```